

ASP.NET Core MVC

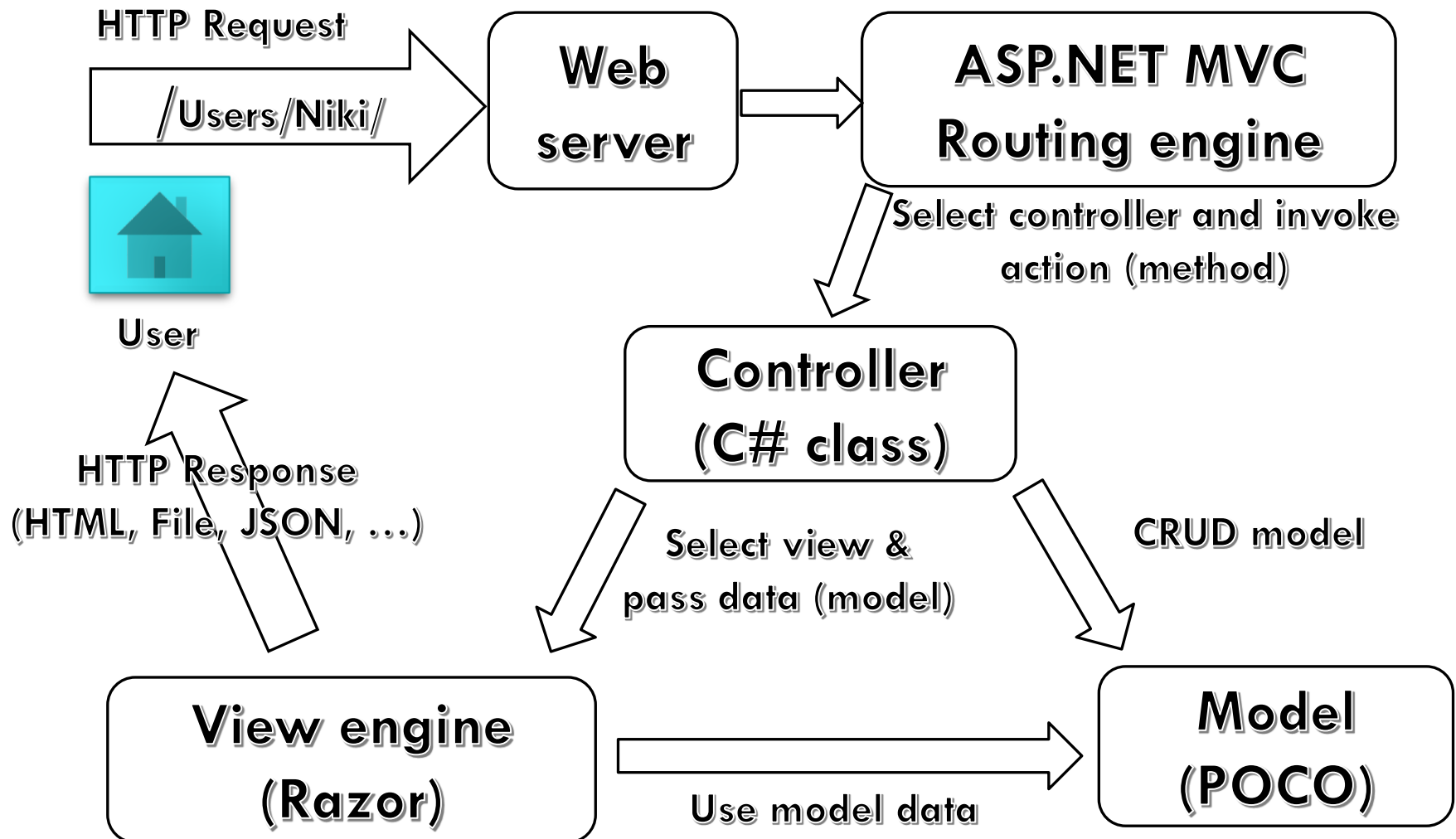
Introdução 2

Programação Visual

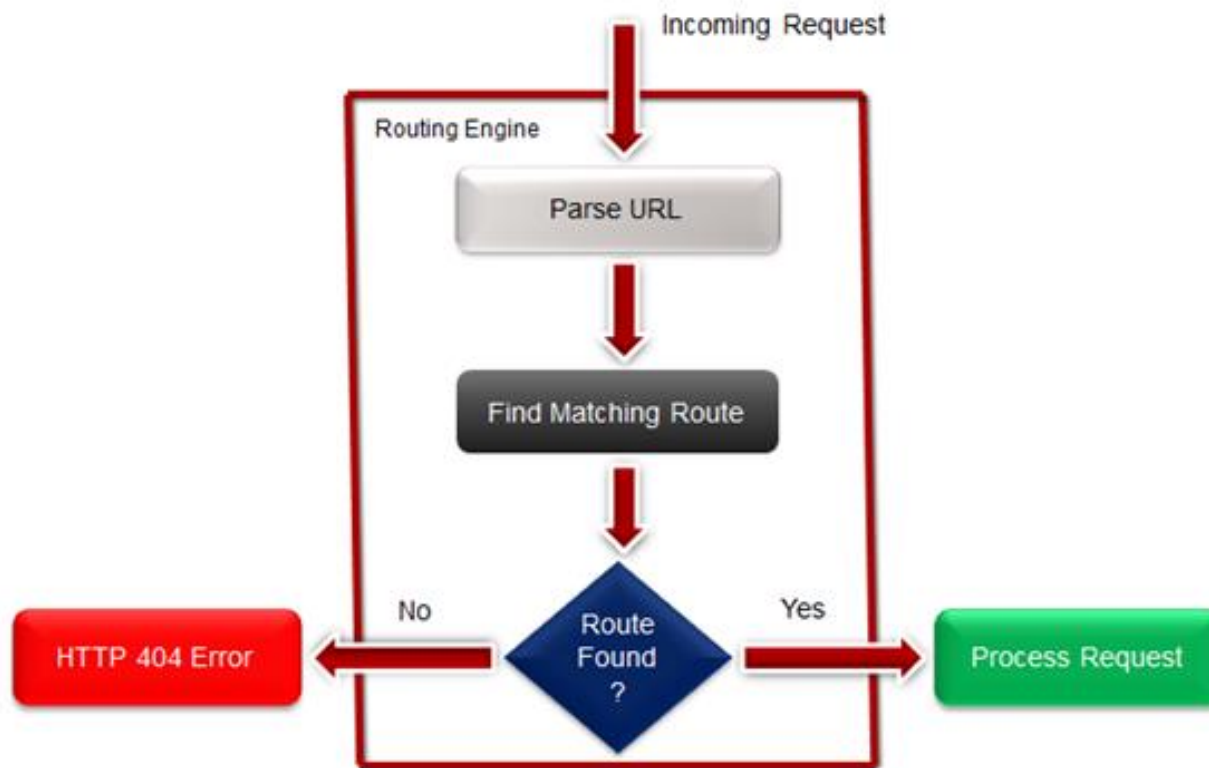
Sumário

- Ciclo de vida de um pedido
- Encaminhamento – Regras
- Ações – Seletores
- Utilitários das vistas – HTML Helpers e Tag Helpers
- Vistas – Layout e secções
- Áreas

Ciclo de vida de um pedido



HOW ROUTING WORKS



- Mapeamento entre padrões, uma combinação de **controlador + ação + parâmetros**
- Algoritmo
 - O primeiro *match* ganha
- As regras de encaminhamento são definidas no código do programa principal dentro do ficheiro **Program.cs**

```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

Route name

Route pattern

Configurado em **Program.cs**

```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

Route name

Route pattern

Default parameters

```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller}/{action}/{id?}",  
    defaults: new { controller = "Home", action = "Index" }  
);
```

<http://localhost/Products/ById/3>

controller: Products

action: ById

id: 3

```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller}/{action}/{id?}",  
    defaults: new { controller = "Home", action = "Index" }  
);
```

<http://localhost/Products/ById>



controller: Products

action: ById

id: (parâmetro opcional)


```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller}/{action}/{id?}",  
    defaults: new { controller = "Home", action = "Index" }  
);
```



<http://localhost/Products>

controller: Products

action: Index

id: (parâmetro opcional)

```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller}/{action}/{id?}",  
    defaults: new { controller = "Home", action = "Index" }  
);
```

<http://localhost>

controller: Home

action: Index

id: (parâmetro opcional)

```
app.MapControllerRoute(  
    name: "users",  
    pattern: "users/{username}",  
    defaults: new { controller = "Users", action = "ByUserName" }  
);
```

```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller}/{action}/{id?}",  
    defaults: new { controller = "Home", action = "Index" }  
);
```

<http://localhost/users/Jose>

controller: Users

action: ByUserName

username: Jose

```
app.MapControllerRoute(  
    name: "users",  
    pattern: "users/{username}",  
    defaults: new  
    {  
        controller = "Users",  
        action = "ByUserName",  
        username = "DefaultValue"  
    }  
);
```

<http://localhost/Users>

controller: Users

action: ByUserName

username: DefaultValue

```
app.MapControllerRoute(  
    name: "users",  
    pattern: "users/{username}",  
    defaults: new  
    {  
        controller = "Users",  
        action = "ByUserName",  
    }  
);
```

<http://localhost/Users>

Result: 404 Not Found

Restrições (**Constraints**) são regras sobre os segmentos de URL

As restrições podem ser fornecidas como expressões regulares (compatíveis com a classe **Regex**)

Definidas como um dos parâmetros de **MapControllerRoute(...)**

```
app.MapControllerRoute(  
    name: "blog",  
    pattern: "{year}/{month}/{day}",  
    defaults: new { controller = "Blog", action = "ByDate" },  
    constraints: new { year = @"\d{4}", month = @"\d{2}", day = @"\d{2}" }  
);
```

O elemento central do padrão MVC;

Devem estar sempre numa pasta com o nome **Controllers**;

Por convenção os nomes dos controladores devem ter o sufixo **Controller**: **"NameController"**;

Os Routers instanciam um controlador por cada pedido:

- ◆ Todos os pedidos são *mapeados* para uma ação específica.

Todos os controladores herdam da classe **Controller**

- Fornece o acesso ao **Request** (informação do pedido)

Ações representam o destino final dos pedidos:

- São métodos públicos do controlador;
- Não estáticos;
- Sem restrições ao valor de retorno.

As ações retornam normalmente um **ActionResult**:

```
public IActionResult Contact()
{
    ViewData["Message"] = "Your contact page.";

    return View();
}
```


ASP.NET MVC mapeia a informação do pedido HTTP para os parâmetros das ações de diferentes formas:

- **Routing engine** fornece os parâmetros das ações:
 - `http://localhost/Users/NikolayIT`
 - Routing pattern: `Users/{username}`
- **URL query string** fornece os parâmetros:
 - `/Users/ByUsername?username=NikolayIT`
- **HTTP post** pode igualmente fornecer os parâmetros:

```
public IActionResult ByUsername(string username)
{
    return Content(username);
}
```

Action Selectors

1. `ActionName(string name)`

2. `NonAction`

3. `AcceptVerbs`

- `HttpPost`
- `HttpGet`
- `HttpDelete`
- `HttpOptions`
- ...

ActionName

- É um **atributo** que permite definir um nome diferente para a ação em vez do nome do método.

```
public class StudentController : Controller
{
    public StudentController()
    {
    }

    [ActionName("Find")]
    public ActionResult GetById(int id)
    {
        // get student from the database
        return View();
    }
}
```

Utilização: <http://localhost/student/find/1>

NonAction

- É um **atributo** que diz que determinado método público não corresponde a uma ação.

```
public class StudentController : Controller
{
    public string Index()
    {
        return "This is Index action method of StudentController";
    }

    [NonAction]
    public Student GetStudent(int id)
    {
        return studentList.Where(s => s.StudentId == id).FirstOrDefault();
    }
}
```

ActionVerbs

- É um **atributo** que define qual o tipo de pedido http a que o método responde. Por omissão responde a pedidos HttpGet.

Método Http	
GET	Para obter informação do servidor.
POST	Para criar um novo recurso
PUT	Para atualizar um recurso existente
HEAD	Idêntico a GET mas o servidor não retorna o corpo da mensagem
OPTIONS	Pedido de informação sobre as opções de comunicação suportadas
DELETE	Para remover um recurso existente
PATCH	Para atualizar completamente ou parcialmente um recurso

ActionVerbs

- É um **atributo** que define qual o tipo de pedido http a que o método responde. Por omissão responde a pedidos HttpGet.

Método Http	
GET	Para obter informação do servidor.
POST	Para criar um novo recurso
PUT	Para atualizar um recurso existente
HEAD	Idêntico a GET mas o servidor não retorna o corpo da mensagem
OPTIONS	Pedido de informação sobre as opções de comunicação suportadas
DELETE	Para remover um recurso existente
PATCH	Para atualizar completamente ou parcialmente um recurso

Respondem a um pedido do Browser;

Implementam a interface **IActionResult**;

São 5 os principais grupos de Action Results:

- Status Code
- Status Code com Object Results
- Redirecionamento
- Ficheiros
- Conteúdo

Através de **ViewData** (dictionary)

- `ViewData["message"] = "Hello World!";`
- `View: @ViewData["message"]`

Vistas **Strongly-typed**:

- Action: `return View(model);`
- View: `@model ModelDataType;`

Através de **ViewBag** (dynamic type):

- Action: `ViewBag.Message = "Hello World!";`
- View: `@ViewBag.Message`

Templates HTML da aplicação

Razor view engine disponível

- View engines executam o código e fornecem o HTML
- Disponibilizam vários *helpers* para a geração do HTML
- Em ASP.NET Core MVC usa-se o *Razor*

É possível passar informação para as vistas através de: *ViewBag*, *ViewData* e *Model* (strongly-typed views)

Views suportam *master pages* (layout views)

Outras vistas podem ser renderizadas (partial views)

Define um modelo comum para o *site*;

Similar às ASP.NET *master pages* (mas melhor!);

Razor view engine renderiza o conteúdo de dentro para fora;

- Primeiro a vista, depois o Layout

@RenderBody() –
indica o local para o
“preenchimento” que as vistas
baseadas neste *layout* têm de
preencher fornecendo o
conteúdo.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
  </head>
  <body>
    <nav>@* Menu *@</nav>
    <div id="body">
      @RenderBody()
    </div>
    <footer>@* Footer *@</footer>
  </body>
</html>
```

As vista não necessitam de especificar o *layout* uma vez que o valor por omissão é dado em `_ViewStart.cshtml`:

- `~/Views/_ViewStart.cshtml` (Código para todas as vistas)

Cada vista pode especificar páginas de layout particulares

```
@{  
    Layout = "~/Views/Shared/_UncommonLayout.cshtml";  
}
```

```
@{  
    Layout = null;  
}
```

É possível ter uma ou mais Secções (**sections**) (opcional)

As secções são definidas nas vistas:

```
@section SideBar {  
    <aside>  
        Some side information  
    </aside>  
}
```

Podem ser renderizadas em qualquer sitio do Layout usando o método **RenderSection()**

- **@RenderSection(string name, bool required)**
- Se a secção for obrigatória e não estiver definida na vista é lançada uma exceção (**IsSectionDefined()**)

View Helpers

A propriedade **Html** das vistas tem métodos que retornam uma *string* que pode ser usada na criação do HTML (**HTML Helpers**)

- Criar inputs
- Criar links
- Criar forms

```
@using (Html.BeginForm("Search", "Users",  
                        FormMethod.Post))  
{  
    @Html.TextBox("username")  
    <input type="submit" />  
}  
@Html.Raw(htmlContent)
```

Existem outras propriedades utilitárias para além da propriedade **HTML**

- **Ajax**, **Url**, custom helpers

HTML Helpers x Tag Helpers

Os **Html helpers** foram substituídos em ASP.NET Core MVC pelos **Tag Helpers**. No entanto, ainda são válidos e são utilizados por detrás dos **Tag Helpers**

Existem alguns **Html Helpers** que não foram substituídos por **Tag helpers**.

Recomenda-se a utilização de **Tag Helpers** dado que estão alinhados com a sintaxe do **Html** e são mais simples de compreender e utilizar.

Method	Type	Description
<i>BeginForm, BeginRouteForm</i>	Form	Returns an internal object that represents an HTML form that the system uses to render the <code><form></code> tag
<i>EndForm</i>	Form	A void method, closes the pending <code></form></code> tag
<i>CheckBox, CheckBoxFor</i>	Input	Returns the HTML string for a check box input element
<i>Hidden, HiddenFor</i>	Input	Returns the HTML string for a hidden input element
<i>Password, PasswordFor</i>	Input	Returns the HTML string for a password input element
<i>RadioButton, RadioButtonFor</i>	Input	Returns the HTML string for a radio button input element
<i>TextBox, TextBoxFor</i>	Input	Returns the HTML string for a text input element
<i>Label, LabelFor</i>	Label	Returns the HTML string for an HTML label element

Method	Type	Description
<i>ActionLink, RouteLink</i>	Link	Returns the HTML string for an HTML link
<i>DropDownList, DropDownListFor</i>	List	Returns the HTML string for a drop-down list
<i>ListBox, ListBoxFor</i>	List	Returns the HTML string for a list box
<i>TextArea, TextAreaFor</i>	TextArea	Returns the HTML string for a text area
<i>Partial</i>	Partial	Returns the HTML string incorporated in the specified user control
<i>RenderPartial</i>	Partial	Writes the HTML string incorporated in the specified user control to the output stream
<i>ValidationMessage, ValidationMessageFor</i>	Validation	Returns the HTML string for a validation message
<i>ValidationSummary</i>	Validation	Returns the HTML string for a validation summary message

HTML Helpers definidos pelo utilizador

Escrita de *métodos de extensão* para a classe **HtmlHelper**

- Retornar **string** ou fazer **override** do método **ToString**
- **TagBuilder** faz a gestão do fecho dos “tags” e dos marcadores


```
public static class HtmlhelperExtensions
{
    public static TagBuilder Image(this HtmlHelper helper,
                                   string imageUrl, string alt)
    {
        TagBuilder imageTag = new TagBuilder("img");
        imageTag.MergeAttribute("src", imageUrl);
        imageTag.MergeAttribute("alt", alt);
        return imageTag;
    }
}

@Html.Image("image.jpg", "Just image");
```

Outra forma de escrever os *Helpers*:

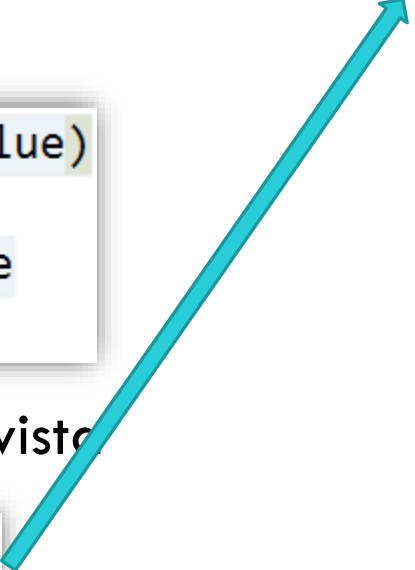
- Criar a pasta **/App_Code/**
- Criar uma vista dentro dessa pasta (por exemplo **Helpers.cshtml**)
- Escrever um helper usando **@helper**

```
@helper WriteValue(int value)
{
    @:Value passed: @value
}
```



Pode ser utilizado em qualquer vista

```
@Helpers.WriteValue(20)
```



Existe muito código numa vista? => criar helpers

Tag Helpers

- Possibilitam ao código do lado do servidor participar na criação e renderização dos elementos html presents nos ficheiros Razor.
- Os **tag helpers** disponibilizam:
 - Uma experiência de desenvolvimento integrada e próxima do html.
 - IntelliSense para a criação de *markup Html* e *razor*.
 - Uma forma de aumentar a produtividade e contribuir para a criação de código mais robusto, confiável e de mais fácil manutenção.
- **Tag helpers** incluídos:
 - Form, FormAction, Input, Label, Option, Select, TextArea, ValidationMessage, ValidationSummary
 - Cache, Distributed
 - Image, Anchor, Script, Link, Environment, etc.

Exemplos

- **Anchor**

- `@Html.ActionLink("Ir para o inicio", "Index", "Home")`
- `<a asp-controller="Home" asp-action="Index">Ir para o inicio`

- **Form**

```
@using (Html.BeginForm("Edit", "Categories"))  
{  
  
}  
  
<form asp-action="Edit" asp-controller="Categories">  
{  
  
}
```

- **Label**

- `@Html.LabelFor(Model => Model.Nome)`
- `<label asp-for="Nome">Nome</label>`

As vistas parciais permitem renderizar parte da página

- Reutilizam pedaços de vistas
- Html helpers – Partial, RenderPartial e Action

Sub-request

As vistas parciais Razor são igualmente ficheiros .cshtml

```
@using MyFirstMvcApplication.Models;  
@model IEnumerable<UserModel>
```

```
@foreach (var user in Model)  
{  
    @Html.Partial("_UserProfile", user);  
}
```

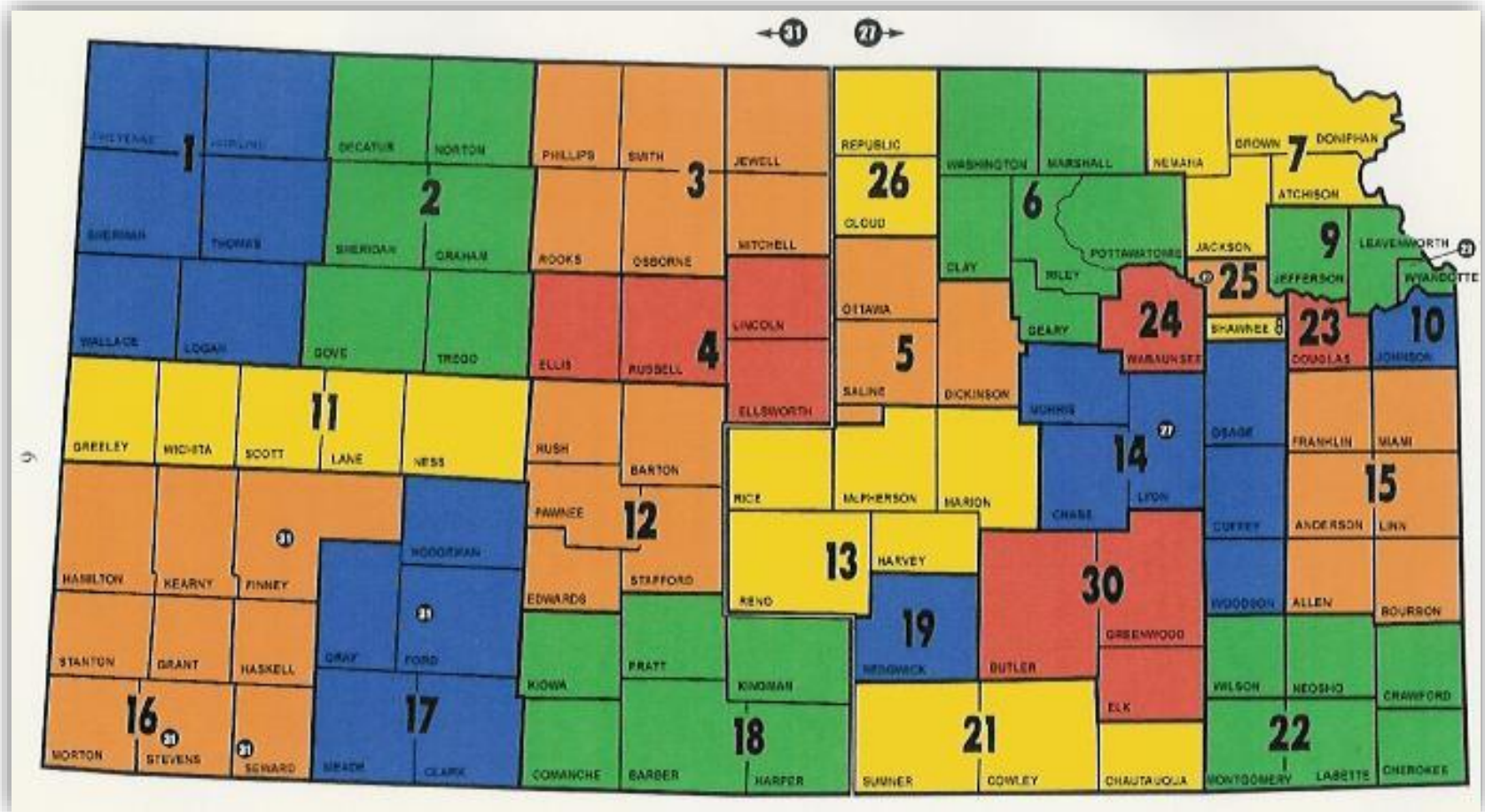
```
@Html.Partial("_UserProfile", user);
```

```
@Html.Partial("_UserProfile", user);
```

```
@using MyFirstMvcApplication.Models;  
@model UserModel
```

```
<h2>@ViewBag.Title</h2>  
<p>@Model.FullName is @Model.Age years old</p>
```

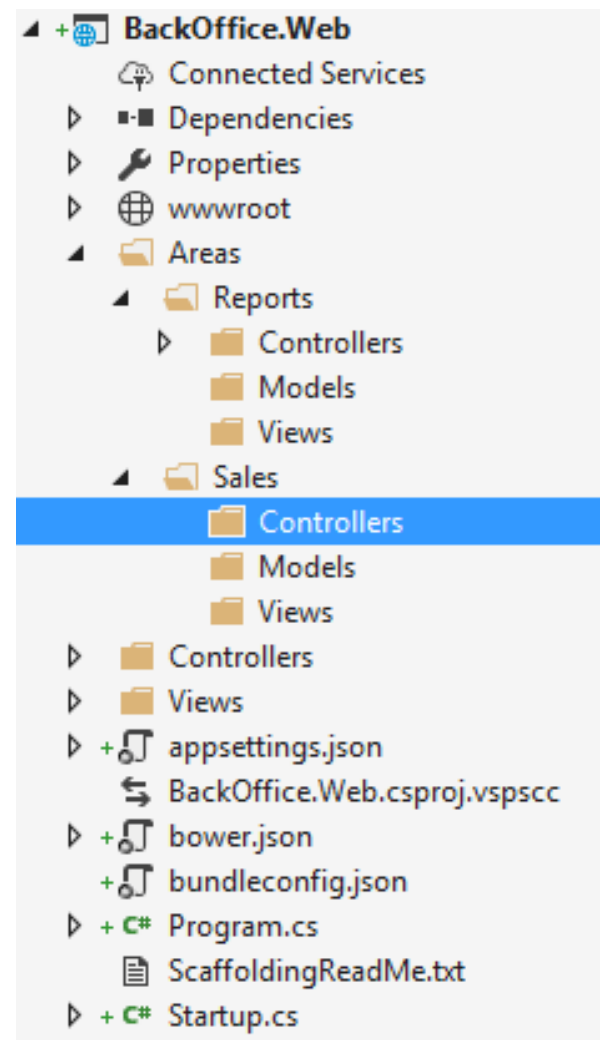
Na mesma pasta que as outras vistas ou na pasta Shared



Algumas aplicações podem ter um grande número de controladores
O ASP.NET MVC permite a partição das aplicações Web em unidades mais pequenas (**areas**)

Uma **area** constitui, na prática, uma estrutura MVC dentro da aplicação
Exemplo: uma grande aplicação de e-commerce

- Loja principal, utilizadores
- Blog, fórum
- Administração



Referências

- ◆ Telerik Software Academy
 - ◆ academy.telerik.com



- <https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0>
- <https://www.tutorialsteacher.com/mvc/action-selectores-in-mvc>
- <https://codingblast.com/asp-net-core-tag-helpers/>